U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology

NISTIR 89-4063

National Computer Systems Laboratory

Hybrid Structures for Simple Computer Performance Estimates

G.E. Lyon

U. S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
National Computer Systems Laboratory
Advanced Systems Division
Gaithersburg, MD 20899

CMRF

COMPUTER MEASUREMENT
RESEARCH FACILITY
FOR HIGH PERFORMANCE
PARALLEL COMPUTATION

March 1989

Partially sponsored by
- Defense Advanced Research Projects Agency
- Bureau of Export Administration
- Department of Energy

Hybrid Structures for Simple Computer Performance Estimates

Gordon Lyon

Advanced Systems Division
National Computer Systems Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

Partially sponsored by
- Defense Advanced Research Projects Agency
- Bureau of Export Administration
- Department of Energy

An abridgment of the text through section 2.3.2 will be given at the International Conference on Computing and Information, **ICCI'89**, *May, 1989, Toronto, Canada, as "Capacity-and-Use Trees for Estimating Computer Performance Variations."*

U.S. Department of Commerce, Robert A. Mosbacher, Secretary

Ernest Ambler, Acting Under Secretary for Technology

National Institute of Standards and Technology
Raymond G. Kammer, Acting Director

March 1989

TABLE OF CONTENTS

Page

Hybrid Structures for Simple Computer Performance Estimates

Gordon Lyon

Even the coarsest performance estimators for a modern computer must account for architectural dependencies and variabilities. For instance, average execution rate is rather sensitive to the match between machine capabilities and application workload.

Computing can be viewed as system components that are subjected to demands of an application, or alternately, as an application workload partitioned by system service. Models based upon this dual perspective help organize simple performance measurements. Several examples demonstrate strengths of a straightforward and flexible partitioning scheme based upon tree graphs. Quite explicit, the graphs promote a more critical view of measurements and support multiple interpretations.

Key words: application; architecture; benchmarks; components; models; performance.

1. Measurements and Structure

Performance measurement of the modern computer is notoriously elusive, its experimental approaches often torn between two extremes. The first summons a large battery of comprehensive benchmark tests, each reflecting the application world [15]. Unfortunately, these tests may be difficult to characterize, and the ensemble expensive to administer, since applications are exceedingly diverse [10].

A second view, taken here, focuses upon barest architectural features. In essence, benchmark measurements of a machine's various processing modes are attached to a simple model of its architecture. This economy, while rougher, provides a broad, accessible summary of salient facts. Although it is generally agreed that the first approach, with its knowledge of applications, yields the fullest characterization [12], circumstances

---

 Partially sponsored by the Defense Advanced Research Projects Agency, ARPA Task No. 5520, the Bureau of Export Administration, BXA Order No. ITA-87-PAY-226-B, and the Department of Energy, DoE Order No. DE-AI05-87ER25046.

arise for which an architecturally-focused view is better. A large number of machine choices may need preliminary winnowing. Or, the application community may be poorly defined, as it is for machines whose export a government wants to control. In export, only the machine architecture and operating system are definitely known [3].

1.1 Workload Partitions

The general perspective is set in simple terms. There is no explicit provision for either *process* or *job*. A workload of *unordered*, but not necessarily unqualified, operations is partitioned into (disjoint) subsets using a system's architecture as a guide. Then an interpretation is applied to these subsets.

To maintain consistency, the method demands some care. The partition must be consistent with the interpretation. For instance, throughout much subsequent discussion, the chosen interpretation establishes processing times for each subset. Overall time is then the sum of times of all subsets. It follows that operations from various subsets *cannot* overlap in time; to do so destroys the consistency of the summation [2]. Executions within a subset proceed in some undefined manner, serial or parallel, but at *one designated rate*. Clearly, another interpretation rule imposes its own partition constraints.

Applications are defined *logically* at the language level, as in FORTRAN, Pascal, or Ada®. However, this is not to say that identical textual repetitions of language expressions incur the same execution costs. A principal tenet is that such is often not the case; as an example, context may place one instruction in the instruction cache, to be fetched from this fast location, whereas distinct circumstances later have an identical instruction fetched from slower main memory. It is assumed that any machines to be compared run essentially the same logical programs, even though respective machine codes are distinct. The actual description of an application workload comprises frequencies of operations on a given machine, subject to whatever classification a partitioning imposes. This application *signature* is much weaker than stipulating processor streams.

1.1.1 Dependencies, Competitions, Trees. Operations with distinct rates often determine distinct partition subsets. This is true for RISC, vector or parallel machines. One partition might be for serial thread operations, another for multiple-threads. Subsets may be further partitioned, so that there are double-threads, triple-threads, etc. Such additional partitionings are usually local to some architectural detail and have no applicability to other subsets. They are conditional, rather than global in scope. Furthermore, subsets may compete against each other. This is true whenever two or more subsets identify with results that are treated the same in further processing; i.e., a floating point value is the same regardless of its scalar or vector origin. These two points, dependencies and competitions, suggest that the partitioning mechanism should not be

general set operations, but rather, a tree notation. A tree expresses dependent or competing partition choices and admits local, *ad hoc* refinements without obscuring other details unnecessarily.

1.2 Architecture and Coarse Evaluations

Determining important performance aspects may be a straightforward interpretation of the hardware and architecture, but this is not guaranteed. Machines hold surprises in capabilities established not through obvious architectural features, but through synergistic strengths and weaknesses of component groups, including compilers and loaders. On the other hand, one would like to simplify specifications and illuminate major performance characteristics of a machine via standard benchmarks. This endeavor is related to performance modeling, and entails many of the same hazards. It is important that the few emphasized features dominate performance. Some general questions on a machine's fundamental balance and capabilities include:

- size of memories
- processor bandwidths
- i/o capabilities
- memory-to-processor bandwidths
- processor-to-processor communication
- memory-to-memory bandwidth

Hillis claims, with good justification, that the above must be in reasonable balance for a system to warrant serious attention [7]. The list is a good minimal tally, a place to start, but there are other points analogous to arguments made about partitioning. Certain machine capabilities will have importance only in the context of others. Thus length-of-vector is a factor for vector processing, but not for scalar processing on the same vectors. Given the specialized nature of many computing elements, the opportunities for conditional capabilities are great.

Another pivotal execution interaction occurs among operation modes. Otherwise interchangeable results may incur very different costs of computation, depending upon their mode of origin. Distinct workload contexts encourage competitions among operation modes that account for many performance variations.

Operations can be classified as *reduced* or *multimodal*. An operation's actual designation depends upon architecture and implementation. For this reason, modal details require special attention.

> *Reduced operations*. The term *reduced* denotes an operation that behaves more or less the same under varieties of processor and system state. On older machines, operations were often quite predictable. Simple formulae were

even provided by manufacturers to calculate the clock cycles for each instruction. In the terminology, these operations were *reduced*, or single mode.

Multimodal operations. The modern machine may exhibit a range of execution behavior for equivalent results. Examples include: scalar or vector dispatch, instruction cache conditioning, memory-fetch anisotropy. Operation times vary greatly.

1.2.1 The Modality. A modality is a set of modes whose operations can yield equivalent results. The modality forms a *k-alternative, forced choice of modes*. The following table summarizes four common (binary) modalities. The first three sometimes occur on the same machine.

	Competitive Modes	Architectural Focus	Appx. Hardware Differences	Improvement w/ Prudent Use
1	scalar vs. vector	*vector processors*	*peak vector is 4x to 25x faster*	*Monte Carlo trial--none lin. alg.--near peak [6]*
2	random GATHER or SCATTER vs. unit-stride	*memory system, vector operations*	*unit-stride is 2.5x faster (at least)*	*3x to 7x estimated [1], [9], [18]*
3	by-row vs. by-column FORTRAN	*virt. memory subsystem, scalar operations*	*page faults slower by 10^4; source: row refs.*	*columns--30% faster for linear eq. solver w/FORTRAN [11]*
4	messages vs. memory refs.	*loosely-coupled nodes*	*memory refs--50x to 10^3x faster*	*array processor w/mesh: 10x faster [13]*

Table I. Four Modalities and Their Performance Variations

On some architectures, one modality *dominates* all others. This is usually true for scientific machines with scalar and vector capabilities.

Example 1: A dominant competition interpreted. A classic competition occurs on machines that perform either scalar or vector computations. Answers are the same done either way, but since vector processing is four to twenty-five times faster, it is naturally preferred. However, the scalar mode persists because its startup is brief. Not every calculation reduces to linear algebra, which is the essence of the vector viewpoint. Consequently, programs remain mixes of vector and scalar, the ratio depending upon application. To account for this, it is very common [17, 5] to (i) estimate scalar and vector rates via benchmark measurements s and v, and (ii) derive a composite performance estimator $p_{s,v}$ that interprets a scalar-vector partition of workload:

$$1/p_{s,v} = \alpha/s + (1-\alpha)/v, \text{ where } \alpha=\text{"scaiar" fraction}$$

The interpretation rule that $p_{s,v}(\alpha)$ exemplifies is Amdahl's law. Consistent in its use of rate and time, this rule will be used exclusively. Additional terms can be added on the right-hand side for further competitor modes. The only restriction is that right-hand numerators sum to unity. Parameter values *s* and *v* are regarded as "basis" capabilities of pure scalar and vector modes. The example's minimalistic partition only resolves to vectors of one length, or to some mean length.

Numerous investigators suggest competing modes to estimate performance [17, 5, 9]. While usually bimodal, trimodal presentations of benchmark data--such as parallel, vector, and serial--have appeared [5]. These simple partitionings of workload can accept further local refinements through multi-level selections. For instance, rather than just a vector or scalar partition, let scalars have further divisions for by-row or by-column (FORTRAN) fetching, as in Table I. Such a partitioning is depicted in the left tree part of Figure 1. The weighted tree is a macro-level flow model decorated with results from benchmarks. It can

- display crucial assumptions in a compact, quickly surveyed format
- support performance estimates, which are computed from its components

2. Capacity-and-Use Tree

A *capacity-and-use tree, CUT*, is a doubly-weighted tree-graph. The unadorned tree describes a system's dominant architecture, while all nodes and arcs have weights of capacity (an admittance) and use (a frequency). Arc capacity admittances c_i describe a system's component strengths. Capacities are admittances because these can be obtained from benchmarks without correcting constantly for code size. Arc frequency weights f_i define application classes through their demands upon architectural features. CUTs varied on workload frequencies generalize the scalar-vector benchmarking interpolation above. (Unlike many *analytic graph* models [4], time is not explicit.)

CUT arc weights are intrinsic to the stage that an arc represents, whereas node weights are cumulative from the tree root. Arcs from a node represent alternatives, e.g., operations on scalars or vectors, operands via inter- or intra-node communications. *Interpretation* assumes that these alternatives never proceed concurrently, so the CUT must be built accordingly. Let C_w and F_w be capacity and frequency weights of tree node W. Distinguished root node R is such that $C_R=1$ and $F_R=1$: This reflects 100% workload at peak performance. Suppose that a directed arc *wx* from W to X has weights $0<c_{wx}\leq 1$ and $0<f_{wx}\leq 1$, subject to $\Sigma_i f_{wi}=1$. Then

$$C_X = C_W c_{wx}$$

$$F_X = F_W f_{wx}$$

A node with no fanout is a leaf. Each leaf *i* has a frequency weight F_i and a capacity C_i.

Leaf weights provide estimates of performance. If all operations run at peak capacity, the "time" is 1/1=1, i.e., a 100% fraction of code divided by the highest normalized rate. Naturally, common cases are worse than this. Thus, F_i/C_i is the cumulative time (relative to unity) for all computations with attributes that match leaf i; a coefficient of overall system effectiveness against peak is then

$$C_{eff} = [\, F_1/C_1 + F_2/C_2 + \ldots \,]^{-1}.$$

2.1 Hypothetical Vector System XXX

Assume from Table I a hypothetical vector, memory-to-memory System XXX with:

- relative rates: scalar=0.1, vector=1.0
- workload mix: scalar=30%, vector=70%
- relative scalar rates: by-row=0.7, by-column=1.0
- scalar workload mix: by-row=50%, by-column=50%
- relative vector rates: GATHER-SCATTER=0.3, unit-stride=1.0
- vector workload mix: GATHER-SCATTER=50%, unit-stride=50%

XXX at its CUT root (Figure 1) has a peak efficiency of 1, but the leaves yield a true efficiency of 0.194 *relative to the application.* This agrees with everyday experience, which seldom approaches anywhere near peak vector performance. Admittedly, too coarse partitionings may ignore startup delays and other real-life elements, although corrections can be made, either in tree arcs--as *ad hoc* partition refinements-- or in any estimator that interprets leaf values. Any new interpretation terms must be consistent with the partition, however. The tree need not be balanced.

2.2 Discussion

The typical modeler will say, "The CUT is not very accurate. It is too simple." However, the hybrid CUT subsumes benchmarking work, e.g. [3], and thus has at least that accuracy. Furthermore, a hybrid framework avoids some taxing problems attendant to pure modeling, e.g., inaccuracies from lacking or incorrect detail. Much fine detail is *implicit* in test codes. Basis benchmark results that decorate a CUT keep it within the realm of reality; each has, after all, actually been observed. Of course, this has its own abuses. Benchmarks can also assume too much or too little, and thereby fail to catch important details.

CUTs enjoy the flexibilities of their simple formal structure. They need not be complex. As a practical issue, a very complex CUT is probably not in the spirit of the method, which is meant to be quick, coarse and explicit. Also, as CUT arborescences multiply, the demand upon application specification grows. Each added fan-out in the tree needs more application information for its weights. A happy medium will arrive fairly quickly, as gains of accuracy from the model diminish and demands for application parameters rise. An interesting study by Wang *et.al.* [16] statistically demonstrates that among the 24 LFK (Livermore loops) benchmarks, there are but three to five predictive dimensions: A few benchmark scores should characterize a machine that is not too refractory to program.

The position in the tree of a modality, such as scalar-vector, depends upon how dominant and how dependent it is. In Figure 1, the scalar-vector modality is the root fan-out because it is independent and dominant. Beginning the tree with another factor would duplicate scalar-vector fan-outs throughout the structure. Secondary fan-outs in Figure 1 are each dependent modes, but this is not generally true in other systems. Some modalities will be independent of each other.

2.2.1 Other Architectures, Other CUTs. In addition to the three modalities of Figure 1, Table I has a fourth, which contrasts processor-to-processor messages against processor private-memory references. Depending upon these communication choices, executions vary by a factor of 10 [13]. Operand-to-processor communication may be up to three decimal orders of magnitude faster when direct from memory. Thus, disparate communication modes might serve well as a first differentiation in a CUT for SIMD array processor performance.

2.3 Modeling Component Changes

The example of interpolation between scalar and vector benchmarks shows how fixed system parameters can be used to estimate performances for differing types of applications. The application signature is the key to this. Another interesting possibility explores implementation (capacity) changes in the system, holding the application signature constant. The following must hold:

1. The architectural layout is *fixed*, i.e., the underlying tree remains the same.
2. The application workload is also *fixed*, so that frequency weights on the tree do not change. (Cases with load redistribution are discussed afterwards as accuracy tolerances of table entries.)
3. Computational capacities (admittances) **can be modified within limits.** This amounts to varying an implementation via faster components, better subunits, or less expensive, slower pieces. But improvements cannot "amplify" capacity, i.e., exceed admittances of 1.

Whenever circumstances allow the above, each CUT can supply tables of equal-gain performance increments. Essentially, factors of capacity are treated as independent contributions to performance. The challenge in this naive but useful formulation is to find restructured forms of a CUT's weights that yield simple tabular entries. Although distinctions will arise among CUTs and their applications, several general rules seem appropriate:

- Select *base values* of CUT arc capacities to which changes may be made. (Each varied capacity establishes a table column.)
- Preclude compound effects. Let value F_i/C_i at leaf i change *only* via one varying arc capacity.
- Sum those leaf weights descendent from a varied arc capacity to determine its contribution to performance.
- Convert the range of a capacity's performance contribution into an integer table column index by dividing with a suitable scaling term (not necessarily an integer).
- Scale all ranges with identical terms; otherwise table columns will not be equi-increment. Incorporate scaling terms into the table's interpretation formula.
- Shorten contribution ranges for which the scaling division is not integral by limiting slightly the corresponding capacity changes.

The spirit and form of the transformations are captured by an example.

2.3.1 A Tabular Format for XXX. Let the system XXX of Figure 1 be the base. Its (relative) times are computed from the leaf entries; they are then adjusted so that the computed coefficient is 1. Simply multiply each contributed time by the actual efficiency coefficient. Thus, from leaves in Figure 1,

"t(A)" =(.15/.07)*0.194=0.415
"t(B)" =(.15/.1)*0.194=0.291
"t(C)" =(.35/0.3)*0.194=0.227
"t(D)" =(.35/1)*0.194=0.068

The interpretation "Rate relative to XXX" is then

$$R_{\text{rel-to-XXX}} = 1/\Sigma_i t(i) = 1/1 = 1,$$

as expected for *base values*.

Factors A (by-row) and C (GATHERing) will constitute a small tableau. If degraded capacities were of interest for B (by-column) and D (unit-stride), these could be included as well. Intermediate Table II depicts a range of contributed times for A and C. Base times result from XXX as the base system. Best times are when the capacities are 1, i.e., fully effective. These two ranges, rounded to 0.12 and 0.16, determine a scaling term, 0.04, that gives reasonable-sized table increments. Other scaling terms yield different table resolutions. Because base values t(A) and t(C) are larger and their improvements

smaller (cf., Table II), actual improvement entries are *decrements*.

	Base	Best	Range	Range, in Increments
A	0.415	0.291	0.124	3 [*0.04=0.12]
C	0.227	0.068	0.159	4 [*0.04=0.16]

Table II. Dividing Contribution Ranges by 0.04 for Increments

The interpretation rule for Table III reflects use of a scaling term:

$$R_{\text{rel-to-XXX}} = 1/[t(i) \text{ changes } +1] = 1/[0.04*(A+C)+1] = 100/[4*(A+C)+100]$$

Circumstances will dictate linear reformulations appropriate to other CUTs.

$R_{\text{rel-to-XXX}} = 100 / [\, 4 * (A + C) + 100 \,]$		
	A: column references	**C: GATHERs**
-4	*n.a.*	fantasy "GATHERer"
-3	huge real memory	better loader and memory
-2	larger memory	faster memory
-1	...	smarter loader
0	*System XXX*	*System XXX*
1	...	...
2	less real memory	clustered references
3	...	...
4	pinch on memory	"hot spot" in GATHERs

Table III. Performance Influence of Circumstances for A and C

The expression in Table III provides an index of computation speed for a new machine variant *relative to the base implementation* of the understood, fixed architecture XXX running *the chosen application.* This simplification is especially useful whenever one application is prominent, preferably dominant, in an environment. (Money estimates for subcomponent substitutions further improve the method's utility.) Suppose there is a machine like XXX, but with a huge amount of real memory (A=-3). Unfortunately, its loader produces clustered references (C=+2). The performance of this "XXX?" machine relative to XXX *and the application* is 100/[(-3+2)*4+100]= 1.04, which hardly seems to justify its higher memory costs. An improved loader would probably make a more competitive product.

2.3.2 Related Work. The tableaux work not only for digital computer modeling, but serve equally well for other engineering practice, such as simplified aerodynamic drag coefficient estimation [19]. Wind tunnel testing is essentially an analog method of deriving aerodynamic information. Two decades ago, White [19] published a short note on an estimation technique for the drag coefficient of automobiles. While construction details are lacking in his communication, it is clear that the method works because (1) automobiles are assumed fixed in architecture (hood, cabin, trunk), and (2) travel is set at highway speed, so the Reynolds number, which determines classes of airflow, is constant across comparisons. Variations are illustrated well by sample calculations for windshield shape: add +1 for full wrap-around, +2 for wrapped ends only, +3 for bowed and +4 for flat. Furthermore, add +1 for an upright windshield, and +1 for rain gutters. Multiply by 0.0095. This contribution of the windshield is added to a base-form drag of 0.16, which would be a teardrop shape with wheels, but otherwise undetailed. Clearly, extending this method to trucks demands new base and additive values, as well as corrections. This says, of course, that the automotive architecture would change. Similar application corrections might be needed to account for much elevated autobahn speeds that have different flow patterns.

White's tables are generally accurate to ±7 %. Some of this uncertainty must surely arise from mutual airflow interference among choices in his tables. The analogy for computer systems is workload redistribution.

2.3.3 When Load Redistributes. In many circumstances, the partition on the workload changes as capacities are varied. For instance, suppose that a machine has vector, scalar, and overlapped scalar-vector modes. Any change to a new scalar performance, *s-new*, affects the overall workload fraction that is overlapped. This is handled by calculating both best and worst redistributions, by entering "s-new" into the scalar unit's column at an index location that reflects a performance midpoint between worst and best, and by declaring the predictive precision of the entry. This is reasonable, since column indices are linked to performance increments, and not to capacities *per se*. Overall table accuracy is established by the entry with worst predictive precision.

Interactions among multiple changes will further degrade the precision of table predictions. Whenever multiple perturbations become too obscuring, a table should be restricted to *one-at-a-time excursions of capacity* from the base set, *i.e.*, "pick a column." Because the restricted table can still be used to compare respective gains of system component changes, the restriction is not severe; a succession of tables might be employed for upgrades over time. Workload rebalancing for a single component change can be far less disturbing than one might expect, although actual tolerances can only be determined case-by-case. Redistributions for a scalar-vector machine are illustrative.

Overlapped Scalar-Vector. The assumed architecture has a scalar mode, a single vector mode, and an overlapped, non-interfering scalar-vector mode. Perhaps this is idealistic, but the example shows well the dual calculations that establish accuracies of table entries. Application fractions and base machine capacities are:

α at C_α (scalar)

β at C_β (vector)

$\alpha'+\beta'$ at $C_{\alpha+\beta}$ (overlapped), subject to $\alpha' C_\beta = \beta' C_\alpha$

Let a new scalar rate be $C'_\alpha = k C_\alpha$. It is as if a k-faster unit had been acquired for the machine.

(i) Worst case. Assume all available places for scalar dispatches were in use in the overlapped partition. Consequently, a faster scalar unit diminishes that portion of the workload done as overlapped, which is the fastest rate. Table IV shows new distributions of load. Note that the overlapped mode has lost $(k-1)\beta'/k$.

(ii) Best case. Suppose there are ample opportunities to dispatch new overlapped scalar operations, that the prior limit on scalar overlapped operations has been the speed of the (now faster) scalar unit (Table IV). Non-overlapped scalar execution drops by $(k-1)\alpha'$.

Identical arguments apply to improved vector capability, but roles of α's and β's interchange. Speeding up an already fast unit is not generally economical, as calculations will illustrate.

	Scalar	Vector	Overlapped
Base	α	β	$\alpha'+\beta'$
(i) Worst	α	$\beta+(k-1)\beta'/k$	$\alpha'+\beta'/k$
(ii) Best	$\alpha-(k-1)\alpha'$	β	$k\alpha'+\beta'$

Table IV. Load Redistributions with k-Faster Scalar

(iii) Sample calculations. Actual figures can sometimes be more revealing than algebraic expressions. Assume a machine as in Table IV such that

$R_{scalar} = 1$ (absolute scalar rate)

$R_{vector} = 10$ (absolute vector rate)

$R_{peak} = 11 = 1 + 10$ (absolute peak rate)

$\alpha = 0.2$, $C_\alpha = 0.091$ (relative scalar rate)

$\beta = 0.6$, $C_\beta = 0.909$ (relative vector rate)

$\alpha' + \beta' = 0.018 + 0.182 = 0.2$ (overlapped fraction)

$C_{eff} = \mathbf{0.327} = (0.2/0.091 + 0.6/0.909 + 0.2/1)^{-1}$

Some revealing numbers emerge from single-component variations. Improving the scalar unit boosts efficiency relative to both the base machine and an improved vector variant (vector rate: 20), the latter performing less efficiently than the base machine. For the given situation, doubling the scalar rate is much more effective. One cannot say without further information whether costs of this doubling are reasonable. The effect of load redistribution is insignificant.

	C_{eff}				$C_{eff}*R_{peak}$
	Worst	*Best*	**Mean**	**Tolerance**	*Mean Absolute*
Base Machine	*	*	**0.327**	*	*3.60*
2x Faster, Scalar	*0.468*	*0.492*	**0.480**	**2.6%**	*5.76*
2x Faster, Vector	*0.192*	*0.200*	**0.196**	**2.0%**	*4.12*
2x Faster, Both	*	*	**0.349**	**6.7%**	*7.68*

Table V. Overlap: Base and Redistributed Performances

Several observations help explain performance changes that accompany single-component variations. Doubling the scalar rate (to 2) causes little shift in the pure scalar fraction for the best case, and none for the worst. On the other hand, shifts in load between vector and overlapped vector involve only moderate changes of rate (10 versus 12). Consequently, performance changes for either best or worst cases are dominated by improvement in the bottleneck scalar mode. Improvements to vector capacity (to 20) hardly affect the pure scalar fraction. Vector and overlapped vector modes undergo large shifts in workload partition between best and worst redistributions, but vector and overlapped vector rates are 20 and 21, a difference that barely matters.

The last variation has both scalar and vector components made 2x faster; this is actually the base machine again, although with a twice higher R_{peak}. Using a linear model and Table V, an improved scalar component will boost C_{eff} by (0.480-0.327) = (0.153). The vector change decreases C_{eff} by (0.327-0.196) = (0.131). The net predicted change in C_{eff} by linear combination, is 0.153-0.131 = 0.022, or 6.7% of 0.327, C_{eff} of

the base machine. This is prediction error, since making all components uniformly faster should not change C_{eff}; it is C_{eff}*Peak that rises. Nonetheless, ±6.7% is a serviceable accuracy.

2.4 Common Leafy Subtrees

It is not unusual that a CUT has common subtrees. In these cases, parts of the tree can be shared. Only leafy subtrees, those whose arcs terminate as leaf nodes, are considered. Other embedded common subtrees can also be merged, but this is counterproductive; the merged subtree has node fan-outs whose arcs can be totally unrelated choices. This only detracts from a display of performance factors. The sharing is depicted in (i) and (ii) of Figure 2.

An independent factor produces duplications in a CUT that arise from a common immediate ancestor node. Identical subtrees arise because the factor presents choices that affect overall performance, but the choices do not condition (i.e., change) contributions from other factors. Such common subtrees will sometimes combine nicely to yield a graph that is simpler, but is no longer quite a tree. See (iii) and (iv) of Figure 2 for examples.

2.4.1 Composite Frequency. Let the frequency weights on two arcs, *a* and *b*, leading to two identical but distinct leafy subtrees be f_a and f_b. The arcs originate from nodes A and B, which have node frequency weights of F_A and F_B, respectively. The frequency weight for the root, R, of a shared subtree T (see Figure 2-ii) is

$$F_R = F_A f_a + F_B f_b.$$

Whenever A and B are the same node (as with an independent factor), $F_A = F_B$, so that

$$F_R = F_A (f_a + f_b),$$

An interpretation is to imagine a single arc from A to R with a composite frequency weight of $f_a + f_b$ (Figure 2-iii). This view preserves a strict tree representation, although a colleague has remarked that it sacrifices some presentation clarity; compound weights on arcs are not obvious.

2.4.2 Composite Capacity. Capacity C_R at the shared root node R is a composite that in essence preserves all time costs of the separate partitions (original subtrees). Equating "new times" = "old times",

$$F_R C_R^{-1} = (F_A f_a + F_B f_b) C_R^{-1} = F_A f_a (C_A c_a)^{-1} + F_B f_b (C_B c_b)^{-1}$$

Solving for C_R,

$$C_R = C_A C_B c_a c_b (F_A f_a + F_B f_b)[C_B c_b F_A f_a + F_B f_b C_A c_a]^{-1}$$

Given the case that nodes A and B are identical, $F_A = F_B$, but $c_a \neq c_b$, since a factor is introduced only when it causes some variability. Then

$$C_R = C_A[c_a c_b (f_a + f_b)/(c_b f_a + c_a f_b)]$$

The effective capacity of a single imaginary link from A (≡B) to R is the right hand expression in square brackets.

3. A Final Example

Having examined several CUTs in the exposition, the reader may want to see what a real one looks like. The sources for this final tree are an *ad hoc* NIST advisory committee on benchmarking for export control, and another group of statisticians at NIST-Boulder. Each has written a report on their work [3, 16]. Their conclusions reinforce each other from rather different perspectives, the first using typical benchmarking design, the second applying statistics to observed benchmark results. The class of machines is vector processors.

Wang, Gary, and Iyer subject data from the 24 Livermore loops, run in 2 modes over 48 systems, to rigorous statistical analyses. A predictive analysis reveals that performance variances in the data are explained by

- Whether a benchmark runs fast or not.(!) *This accounts for 92% of variation.*
- Whether a system has vector capability.
- Vector length. *These three cover 98% of variation in LFK (Loops) data.*

This first set of measurements shows that results on but one dimension, such as Linpack's peak vector measurements, cannot alone explain performance variance [16]. However, obvious interpretive deficiencies in the first principal component lead to another test. A cluster analysis separates the observations into groups distinguished as (i) scalar, (ii) peak vector, or (iii) moderate vectorizability, in character. Combining analyses, important aspects are

1. scalar rate
2. peak vector rate
3. rate for intermediate-length vectors

4. compiler vectorization capability

A NIST advisory committee [3] had earlier recommended benchmarks for aspects 1-3 for assessment of exported vector machines. Since point 4, degree-of-vectorization, is actually determined by program and compiler, the corresponding CUT (Figure 3) subsumes this aspect in its arc weights. Hence, the two approaches--architectural and statistical-- dovetail perfectly. In addition, the treatment of vector lengths is very much consonant with Hockney and Jesshope [8]--capacities on the vector subtree can be approximated by their maximum performance, r_∞, and half-performance length, $n_{1/2}$. Figure 3A is appropriate if overlapped scalar-vector is possible *and important.* In either Figure 3 or 3A, the exact number of arcs for vectors of various lengths is determined by *the required resolution of the model.* A very coarse model will have arcs only for (i) near peak, (ii) a mid-range around $n_{1/2}$, and (iii) a slower performance for shorter vectors. More arcs for finer vector partitioning will improve predictions, but also require more detail about application workloads.

The committee recommends that no single figure be derived from their benchmarks (or here, the CUT of Figure 3). In this light, various leaves of the CUT have their own interpretations. Tailored to special requirements of export control, this view may be inappropriate for other applications of Figures 3 and 3A. Fortunately the war horse, Amdahl's law, will work with the partitioning, so ordinary estimates of average performance can be made as well.

4. Summary and Conclusion

Coherent performance summaries constitute a major problem for modern-architecture computers. For example, any average execution speed must be carefully qualified. Even such coarse performance evaluations must account for dependencies and variabilities within the architecture.

System components are loaded by demands of an application, or alternately, an application workload is partitioned by system service. Models based upon this dual perspective help organize measurements to provide simple performance estimates. Several examples have shown the strengths of a straightforward and flexible partitioning scheme based upon tree graphs. Quite explicit, the graphs support multiple interpretations and promote a more critical role for measurements.

The capacity-and-use tree (CUT) is a natural partition mechanism decorated by corresponding benchmark results. Its strengths are explicitness and malleability; a CUT accommodates architecture, implementation and application within a single compact structure. Parametric variations on the structure yield very compact tableaux that emphasize equal-gain increments: entry accuracies reflect the best and worst of workload redistributions. The tableaux resemble those in other engineering practices, such as charts for estimating coefficients of aerodynamic drag.

The CUT subsumes other simple flow models. Decorated with measurement results, it is a mildly formal, compact declaration of perceived influences. The CUT is attractive as a structure for reporting gross performance characteristics of a machine.

4.0.1 Acknowledgment. Thanks to Robert Carpenter and Carl Smith for questioning numerous points in earlier versions of the text.

5. Citations

[1] *Private conversations with D. Bailey and J. Gary.* The range combines two informal estimates, one of 3x and the other a range of 3.5x to 6x. These numbers should not be taken too seriously, but the effect of SCATTER-GATHER practices should. van Waveren's article [18] has some nice examples on the usefulness of SCATTER-GATHER.

[2] G.M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proc., AFIPS Spring Joint Computer Conference 1967*, Atlantic City, NJ, April, 1967, 483-485.

[3] D. Bailey, E. Brooks, J. Dongarra, A. Hayes, M. Heath, and G. Lyon, "Benchmarks to supplant export FPDR calculations," NBSIR 88-3795, June 1988, 20pp.

[4] J.C. Browne, and A.K. Adiza, "Graph structured performance models," in **Performance Evaluation of Supercomputers**, J. Martin (ed.), Elsevier Science Publishers B.V., 1988, 239-281.

[5] I.Y. Bucher, "The computational speed of supercomputers," Report LA-UR-84-740, Los Alamos National Laboratory, Los Alamos, New Mexico, 1984, 15pp.

[6] J. Dongarra, "Performance of various computers using standard linear equations software in a FORTRAN environment," Technical Memorandum (issued periodically), Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 20pp.

[7] D. Hillis, Unpublished remarks on architectural similarity, given at the 12th Annual Int. Symp. on Computer Architecture, Boston, MA, June, 1985.

[8] R.W. Hockney, and C.R. Jesshope, **Parallel Computers.** Bristol, England: Adam Hilger Ltd., 1984.

[9] O.M. Lubeck, "Supercomputer performance: the theory, practice, and results," in **Advances in Computers, Volume 27**, M.C. Yovits (ed.), Boston, MA: Academic Press, Inc, 1988, 309-362.

[10] G.E. Lyon, "Design factors for parallel processing benchmarks," to appear in *Jour. of Theoretical Computer Science,* (April 1989).

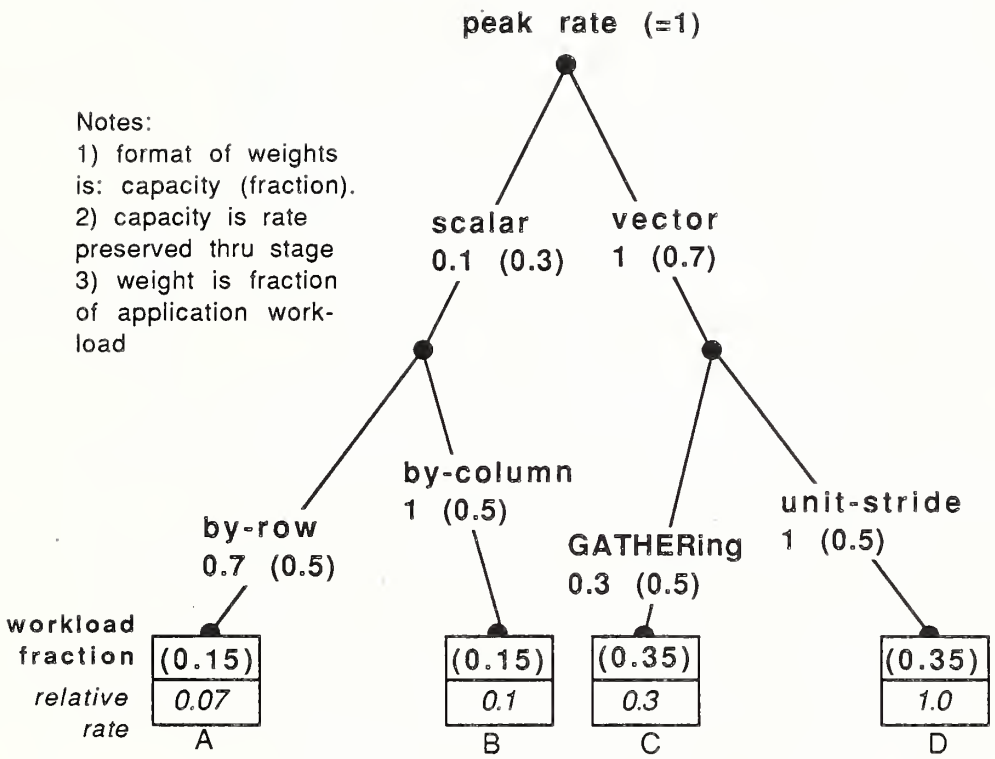
[11] C. Moler, "Matrix computations with FORTRAN and paging," 268-270, *Comm. ACM 15*, 4(April, 1972).

[12] D. Potier, "Analysis of determinant factors for the performance of vector machines," in **Supercomputing**, A. Lichnewsky and C. Saquez (eds.), Amsterdam: Elsevier Science Publishers B.V. (North-Holland), 221-236.

[13] S.F. Reddaway, "Achieving high performance applications on the DAP," in *Proc., CONPAR 88, Stream 'A'*, Brit. Comp. Soc., 1988, 233-241.

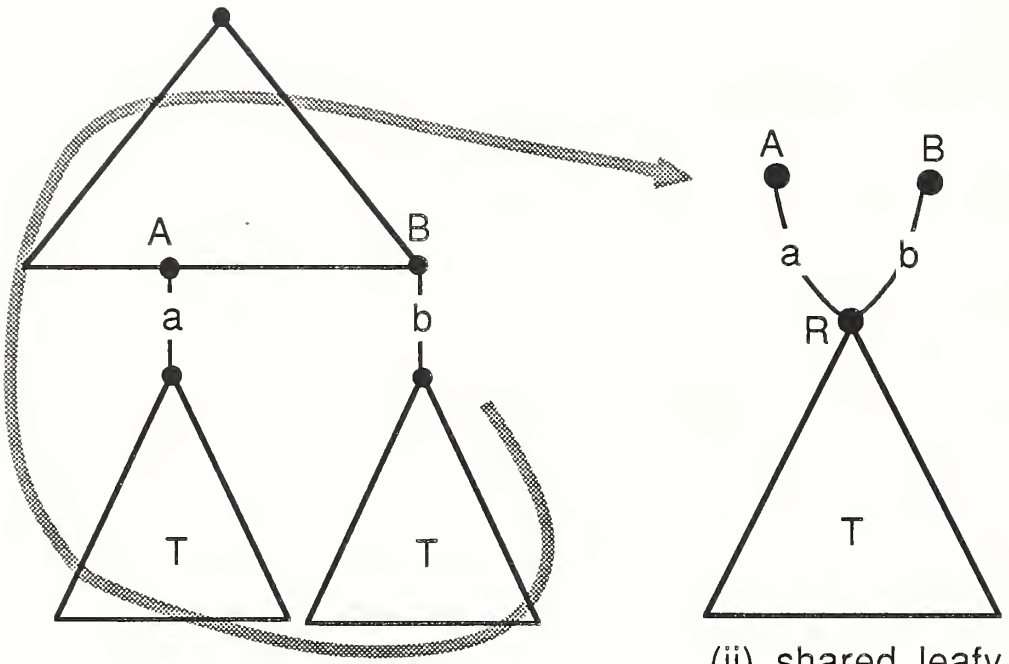
[14] A.P. Reeves, and M. Gutierrez, "On measuring the performance of a massively parallel processor," in *Proc., Int. Conf. on Parallel Processing*, August, 1988,

261-270.
[15] A.J. van der Steen, "Proposals for standard benchmark programs for supercomputers," in *Proc., CONPAR 88, Stream 'C'*, Brit. Comp. Soc., 1988, 58-66.
[16] J.C. Wang, J.M. Gary, and H.K. Iyer, "On the analysis of computer performance data," Draft paper, NIST, Dec. 1988, 33pp.
[17] W.H. Ware, "The ultimate computer," *IEEE Spectrum*, 84-91, (March, 1972).
[18] G.M. van Waveren, "Application of sparse vector techniques on a molecular dynamics program," in **Algorithms and Applications on Vector and Parallel Computers**, te Riele, H.J.J. *et.al.* (eds.), Amsterdam: Elsevier Science Publishers B.V., 1987, 405-428.
[19] R.G.S. White, "Rating scale estimates automobile drag coefficient," *SAE Journal 77*, 52-53, 6(June, 1969).



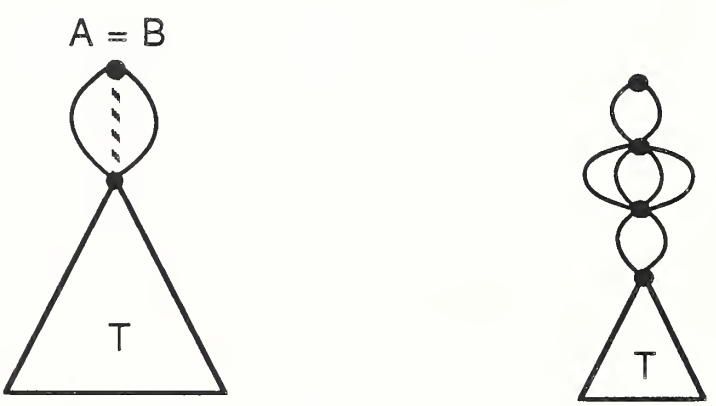
$$C_{effic} = [\ .15/.07 + .15/.1 + .35/.3 + .35/1]^{-1} = 0.194$$

Figure 1. CUT Diagram for Hypothetical System XXX



(i) common leafy subtrees

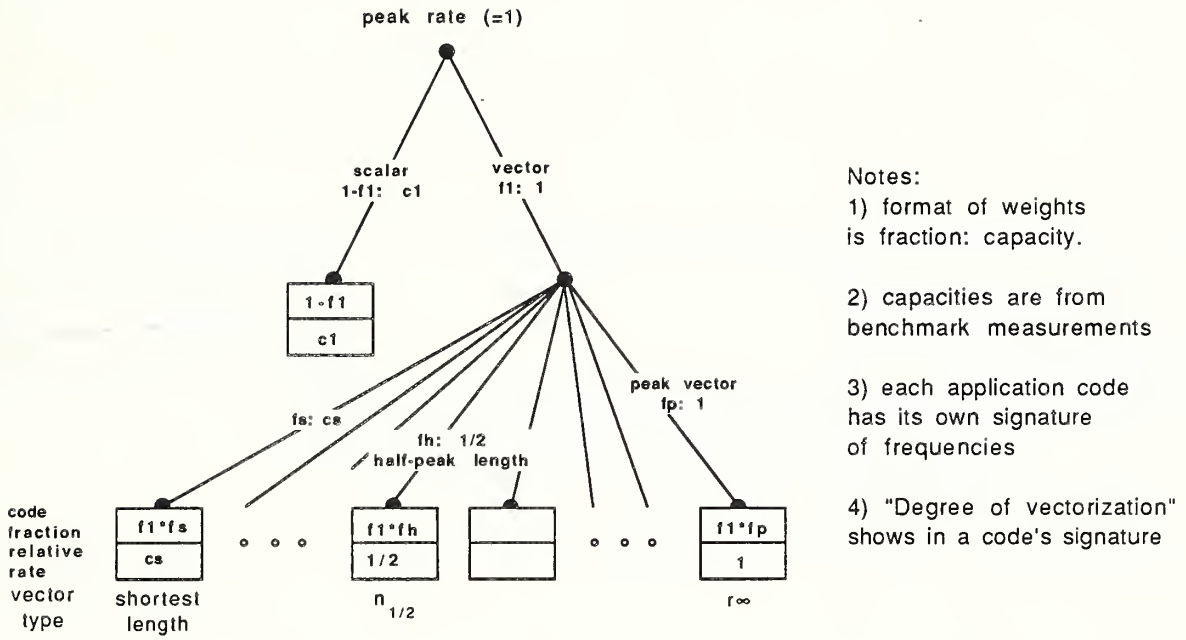
(ii) shared leafy subtree, T



(iii) an independent factor

(iv) several independent factors

Figure 2. Common Subtrees



Notes:
1) format of weights is fraction: capacity.

2) capacities are from benchmark measurements

3) each application code has its own signature of frequencies

4) "Degree of vectorization" shows in a code's signature

Figure 3. CUT Diagram for Typical Vector System

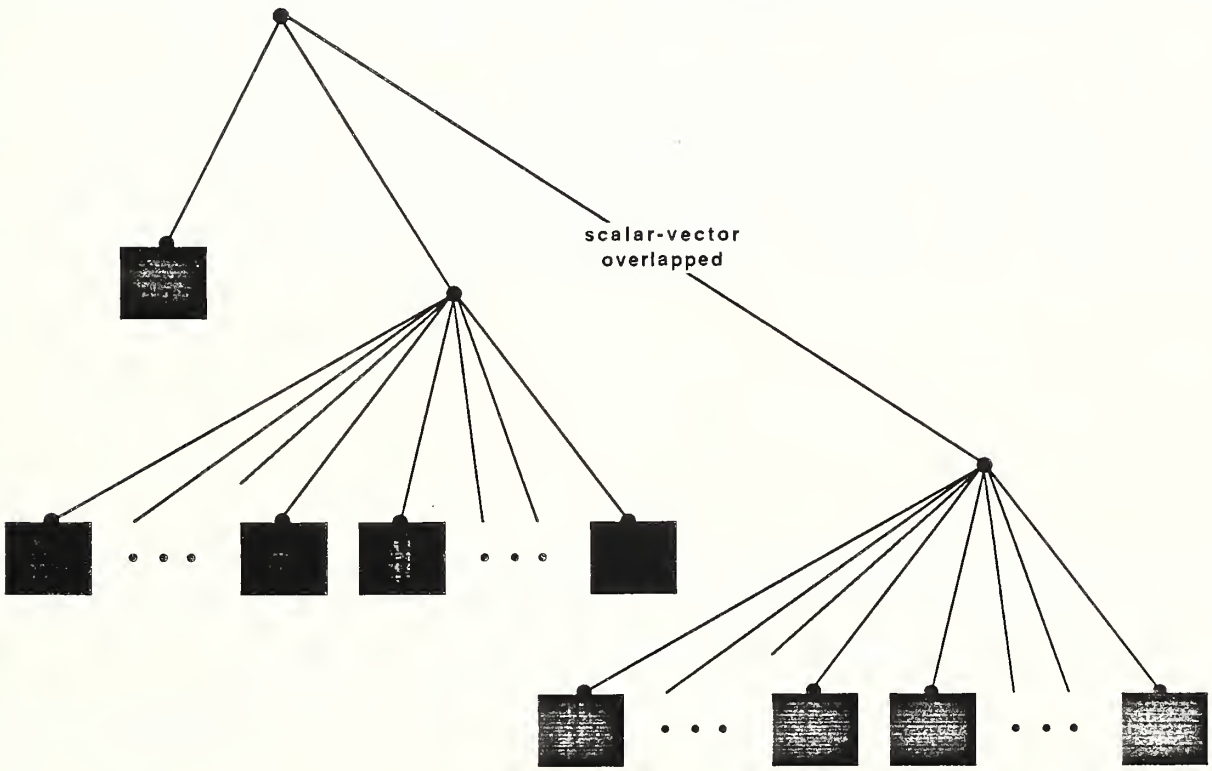


Figure 3A. CUT: Overlapped Scalar-Vector

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NISTIR 89-4063	2. Performing Organ. Report No.	3. Publication Date MARCH 1989

4. TITLE AND SUBTITLE

Hybrid Structures for Simple Computer Performance Estimates

5. AUTHOR(S)

Gordon Lyon

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
U.S. DEPARTMENT OF COMMERCE
GAITHERSBURG, MD 20899

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

Defense Advanced Research Projects Agency, Arlington, VA 22209

Bureau of Export Administration, Washington, DC 20230

Department of Energy, Washington, DC 20545

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

Even the coarsest performance estimators for a modern computer must account for architectural dependencies and variabilities. For example, average execution rate is rather sensitive to the match between machine and application workload.

Computing can be viewed as computational components that are loaded by demands of an application, or alternately, as an application workload partitioned by system components. Models based upon these simple perspectives help organize simple performance measurements. Several examples demonstrate strengths of a straightforward and flexible partitioning scheme based upon tree graphs. Quite explicit, the graphs promote a more critical view of measurements and support multiple interpretations.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

application; architecture; benchmarks; components; models; performance

13. AVAILABILITY

☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

☒ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES 24

15. Price $9.95